COMP1917: 15 File IO

Sim Mautner

s.mautner@unsw.edu.au

October 9, 2016

Purpose

- Read/write external files from within an application.
- Previously, all data had to be entered each time an application was run.
- This lets us save data from the application and retrieve it for later use.

- This lecture covers text files, not binary files. There are different modes and read/write functions for binary files.

Examples

- In a game:
  - Settings file for an application (sound on/off).
  - High scores.
  - Saved games.
  - Data (eg maps) for different levels of a game.
- In a word processor:
  - The data in a text document for a word processor.
  - The saved styles for different types of headings, paragraphs etc that might be used in a different document.
- In a browser:
  - Bookmarks.
  - Autofill usernames and passwords.
- In a calendar:
  - All the events, their times and dates, etc.

File IO in C

- Open a file
- Read/Write to the file
- Close the file

- While a file is ‘open’ it is considered ‘locked’ and cannot be used by any other applications until it has been ‘closed’. Closing a file ‘releases’ the ‘lock’ on the file, allowing other applications to gain access to it.

Opening a File

```
FILE *fp = fopen("filename.txt", "w");
```

- `fopen` - opens a file
- `parameter 1` - the name of the file to be opened
- `parameter 2` - the mode in which to open the file
- `return value` - a pointer to the file which has been opened. This pointer is then used to reference the opened file for operations such as reading, writing, and closing of the file.

File Modes

- `r`
  - Opens an existing text file for reading purpose.
- `w`
  - Opens a text file for writing.
  - If it does not exist, then a new file is created.
  - Starts writing from the beginning of the file.
- `a`
  - Opens a text file for writing in appending mode.
  - If it does not exist, then a new file is created.
  - Starts writing at the end of the existing file contents.
- `r+, w+, a+`
  - Opens a file for both reading and writing.
  - `w+` truncates the file length to zero if it exists, while `a+` starts reading from the start of the file and writing at the end of the existing file contents.

Demo

- Ex 1: Use `fopen` to create a file. Check that the file exists.

Writing/Appending to a File

```
fputs("the text I want to write in the file\n", fp);
```

Or

```
fprintf(fp, "the text I want to write in the file\n");
```

Demo

- Ex 2: Use `fputs` or `fprintf` to write to a file. Check that the text has been written to the file.

Reading from a File

```
char lineOfText[MAX_LINE_LENGTH];
char * success = fgets(lineOfText, MAX_LINE_LENGTH, fp);
```

Or

```
char lineOfText[MAX_LINE_LENGTH];
fscanf(fp, "%s", lineOfText);
```

Demo

- Ex 3: Use `fgets` or `fscanf` to read from the file from Ex 2. Check that the contents is as expected by displaying it.

Closing a File

```
int fileCloseSuccess = fclose(fp);
```

- `fclose` - closes a file.
- `parameter 1` - the file which should be closed. This parameter should be of type `FILE*`.
- `return value` - returns 0 if the close has been successful or `EOF` if there was an error.

- Don't forget to close the file.
- If you forget, it may be unable to be opened by another application, or at by the same application at a later stage because it has been 'locked'.

Exercises

- Ex 4: Write an application which uses the following functions:
  1. Write a function which reads in 10 names and grades from `stdin` into an appropriately designed data structure and returns that data structure.
  2. Write a function which saves the data in your structure into a file called 'grades.txt'.
- Ex 5: Write an application which uses the following functions:
  1. Write a function which reads in 10 names and grades from 'grades.txt' (created in Ex 4) into an appropriate data structure.
  2. Write a function which prints out the data in the structure.